A Model-Driven Approach for System Administration

Marco Centurión, Maximiliano Kotvinsky, Daniel Calegari, Andrea Delgado
Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 11300
{mcenturion, maximiliano.kotvinsky, dcalegar, adelgado}@fing.edu.uy

***Abstract*—System administration requires maintaining network infrastructure and the software running on it. Within an IoT context, it also integrates different devices, communication protocols, and layers. The configuration process is usually carried out by system administrators which perform the configuration manually or through the definition of low-level scripts. Configuration manager tools, such as Puppet, improve this aspect, by adding a level of abstraction from the infrastructure and allowing a better systematization and automation of the configuration process. However, they do not provide a visualization of the whole infrastructure. In this paper, we present a model-driven approach linking visualization and automation needs for system administration. We defined a modeling language for a network infrastructure linking hardware, software and configurations, specified as a UML profile of deployment diagrams and extendable for IoT elements. We generate configuration scripts for Puppet and provide a functional prototype and a case study demonstrating the technical feasibility of the ideas.**

***Index Terms*—System administration, IoT, Model-Driven Engineering (MDE), UML profile, Puppet**

I. Introduction

A system administrator, or sysadmin [1], is responsible for the installation, configuration, monitoring and assurance of an IT infrastructure. This infrastructure is usually composed of computer networks with hardware, such as switches, routers, servers and workstations, and software running on them, such as firewalls, databases and web servers. In the context of the Internet of Things (IoT) [2], [3], [4], as the network of physical things and virtual elements that communicate and interact with each other, other devices have been increasingly integrated such as RFID tags, sensors, actuators, mobile phones, etc. [2].

Configuration management (CM) [5] ensures that the configuration of the IT infrastructure keeps known, updated, stable and trusted. Each component has his state, which may range from a simple network configuration to ensuring that a program is installed or a configuration file is present. In this context, sysadmins need not only to configure each component but also to document the whole network configuration.

In IoT architectures [3] several layers, objects, protocols, and configurations are needed to provide IoT key elements such as identification and tracking technologies, wired and wireless sensor and actuator networks, enhanced communication protocols, and distributed intelligence for smart objects [2]. A hierarchical network is also defined for easing data processing at different levels: Edge (network edge, i.e., devices), Fog (local cloud services) and Cloud (global services), which also needs configuration and monitoring [4], [6].

The configuration process in either context (local, global) is usually done through a low-level script coding process that becomes repetitive and often chaotic. Moreover, documentation is usually scarce and outdated, and thus, CM becomes an expensive and error-prone process due to the continuous evolution and diversity of network components. In this context, Model-Driven Engineering (MDE) [7] techniques can support many CM needs, e.g., domain-specific modeling languages can provide a way to represent different aspects of the network, leveraging a heterogeneous software and hardware infrastructure, and to generate the configuration scripts to be deployed on it. This improves the global comprehension of the IT infrastructure among sysadmins and potentially eliminates the need for setting configuration scripts manually.

There are previous works related to both visualization and automation aspects. Some tools allow to visualize general information of a network (e.g., LanFlow) but without any connection with CM needs. There also are automated configuration tools [8] (e.g., Puppet) which leverage the specification of system configurations and provide a way of applying such configurations against the target network components. Nevertheless, the two aspects have not been completely linked.

In this work, we present a proposal that relates both aspects. In particular, we define a UML profile of deployment diagrams for the specification of aspects of a computer network that links hardware, software, and configurations. We also define model transformations experimenting with the automatic generation of configuration scripts in Puppet. We implemented these aspects within a functional prototype (available at [9]) of a configuration tool, and we developed a case study that puts into practice our solution and shows the technical feasibility of the ideas. Although we are experimenting with system administration in general, our proposal lays the foundations for its application in a more general context for IoT configuration management, extending the proposal with IoT specific elements and configuration challenges.

The rest of this paper is structured as follows. In Section II we present related work concerning basic system administration and MDE. In Section III we present a requirement analysis and refine the scope of this work. Then, in Section IV we introduce our solution and in Section V we describe a functional prototype and a case study. In Section VI we discuss how our basic approach can be extended for considering a more general context for IoT. Finally, in Section VII we provide some conclusions and an outlook of future work.

II. Related Work

There are several tools for the generation of network topologies and their simulation, e.g., [10]. These tools, however, are focused on the structure of the topology at a high level and they do not differentiate particular network devices. There are also tools for the visualization of a network topology, such as LanFlow[1] and SolarWinds[2]. The later also allows the detection of a network, including its topology, network devices, servers, and virtual hosts. Although automatic detection is of interest, these tools do not provide means for configuring the network.

The use of MDE was explored in the context of computer networks. In [11] the authors present an approach for wireless sensor networks modeling. The proposal introduces three levels of abstraction to build: specific domain models, descriptions of component-based architecture, and platform-specific models. Transformations are also defined between these three levels. In [12] the authors present YANG, a data modeling language for the network configuration protocol (NETCONF), allowing a complete description of all data sent between a NETCONF client and server. In [13] the authors develop a tool to standardize network administration to configure any protocol independently of the provider. They define a domain-specific language for the configuration of network devices, without considering the configuration of computers or servers. However, it does not define any form of integration with another tool.

Configuration management was also studied in the context of MDE [14], [15] as a general discipline. Moreover, there are many model-driven proposals for the automation of infrastructure configurations [8]. Puppet[3] provides a declarative domain-specific language to describe system configuration (resources and their state), abstract from platform-specific aspects. A configuration can be either applied directly to the system or compiled into a catalog and distributed to the target system in which an agent enforces the configuration. It also allows centralizing the configuration providing versioning and synchronization features.

III. Requirements for system administration

We identified a set of requirements to apply a model-driven approach for system administration, which we organized in two categories regarding i) the modeling language and generation, and ii) the tool support for the approach.

A. Modeling language and generation requirements

These requirements are summarized in the following:

1) RQ1: Modeling language: The modeling language needs to express the elements comprising the network to be configured, including different types of nodes such as PCs, servers, devices (routers, switches), among others. The language should include these concepts and the corresponding relationships between them, to be able to model the network. It also needs to be flexible enough to allow changes or the introduction of new devices. On the network structure model, we also need to be able to specify the desired configuration for each component, which will define the configuration model.

2) RQ2: Standardization of models: The modeling language should follow standards and/or existing agreements on networks elements definitions and modeling, to be understood, used and integrated with existing models and/or network descriptions.

3) RQ3: Scripts generation: From the configuration model, the configuration scripts should be derived which can be used to configure the devices. To generate the configuration scripts an M2T transformation will be provided, in which knowledge regarding mappings between configuration model elements and script configuration elements is explicit. The generated scripts could be in different formats, i.e., the tool should provide options to generate different types of scripts (e.g., XML, text, specific format for different CMT, etc.)

B. Tool support requirements

Several aspects should be covered:

1) RQ4: Model editor: The editor will implement the modeling language to obtain network structure and configuration diagrams, which can be edited and modified. It will store models in a standard format and provide export functionalities for different formats, to allow integration with existing tools.

2) RQ5: Application of the generated configuration: After the configuration files are generated from the model, it should be applied in an automated and centralized way to be able to install and implement the desired configuration over the target network. This functionality can be achieved by integrating already existing tool support (CMT).

3) RQ6: Distribution of configuration scripts into nodes: Another desirable characteristic is the capacity of sending the generated configuration scripts into the corresponding nodes and devices, to support the centralized configuration of devices. CMTs already provide a centralized server which is in charge of carrying out the necessary actions to configure network nodes, thus we need to integrate our tool with them.

4) RQ7: Existing network detection: The capacity to detect the structure of an existing network and generate a model to represent it could be useful to help in creating the initial model. This functionality becomes more important as the size of the initial topology grows, and could even be essential in cases where the target system or network is of the order of hundreds of devices (especially on an IoT context). Some tools already provide this functionality.

5) RQ8: Detection of network changes: The detection of network and/or device changes could also be important to automatically update the corresponding model. Updating the model in an automated way helps speed up the process of changing the model to reflect changes in the real world, and rapidly generate the necessary scripts to update the corresponding configuration. As with the previous requirement, some tools already provide this functionality.

[1]LanFlow Net Diagrammer: http://www.pacestar.com/lanflow/

[2]SolarWinds: https://www.solarwinds.com/es/network-topology-mapper

[3]Puppet: https://puppet.com/

IV. Proposed solution

We describe a solution and implementation covering requirements RQ1 to RQ6 (presented in Section III). Full solution is available at [9]. Requirements referring to automatically detect a network (RQ7) and its changes (RQ8), as well as an extension of the ideas for a more general IoT context, were left for future work and discussed in Section VI.

A. Modeling language

For the definition of the modeling language we faced two main challenges: i) to analyze the specific domain in order to identify key concepts and relationships that we need to model the network structure, devices, and their configuration, and ii) to decide between developing a DSL or an extension of UML by means of a UML profile. The solution we present in this subsection satisfies requirements RQ1, RQ2, and RQ4.

1) Specific domain analysis: To model the network topology, nodes and their connections should be specified. Nodes can be of two types: physical and logical nodes. Once the physical and logical topology is modeled, the configuration on the desired elements can be specified.

Physical nodes to be modeled include network devices such as routers and switches, servers, personal computers (PC or workstation), and other devices that can be connected to a network such as printers, scanners, etc. These will be modeled with the generic term "devices". `Router` and `Switch`, as network devices, is essential to define the network topology, connecting its elements. `Server` and `PC` must be differentiated since they provide different functions. In the case of logical nodes, i.e., logical elements corresponding to components in physical nodes, key elements to be included are: `Operating system` of `Servers`, and `PC`, the `Firmware` of network devices `Router` and `Switch`. Also, the software installed on each node is of interest, in particular: `Firewall` component, `Runtime` (e.g., `Java`), `Application Server` (e.g., `Tomcat`), `HTTP Server` (e.g., `Apache`), `Database Engines` (e.g., `MySQL`).

Modeling the desired configuration is a key element in our proposal.We identified three main configuration scenarios. `Software configuration` refers to configuring logical nodes, where each type of the component (OS, application server, database, etc.) present specific configuration elements representing their configurable aspects. `Files configuration` refers to specific actions over system files, i.e., verify existence or content, copy and delete files, etc. This configuration is key to detect if a manual configuration of files is required. `Free configuration` is defined to be able to register any type of configuration that cannot be automatically applied by a tool, but can help in manually configuring elements.

A configuration can also be applied to a set of nodes when devices are of the same type and applying the configuration individually requires more effort, so a configuration can be applied to one or many devices. In Fig. 1 we present the main concepts and relationships in the proposed solution.

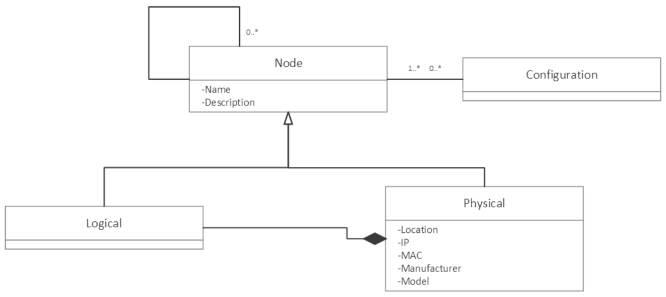


Figure 1: Main concepts and relationships in the solution

2) Domain Modeling Language: We decided to extend UML with specific concepts for our domain, based on the following reasons. UML is a standard language, which is understood and used by many software professionals, provides a base syntax and corresponding semantics that help to understand the extension elements and several tools already exist that provide support for it. Also, the deployment diagrams fit the domain analysis we presented earlier. This kind of diagram shows how different nodes are interconnected in a distributed system, which components are located in which nodes, and implemented by artifacts. `Nodes` are specialized in two elements: `Device` which represents physical machines, and `ExecutionEnvironment` which are assigned to `Device` nodes. `Artifact` elements can be deployed into nodes, and `Deployment` elements represent relationships between artifacts and nodes, as well as characteristics assigned to these relations. Communication links model system connections between elements.

3) UML profile definition: To define the UML profile we extended concepts already provided in the deployment diagram that we presented earlier. Extending `Device` element we represent `Physical nodes`, and extending `ExecutionEnvironment` we represent `Logical nodes`, each with corresponding stereotypes. The parent element `Node` is not extended since the base element already provides the information needed. To represent a `Configuration` component we extend the `Artifact` component with the corresponding stereotype. This extension with main concepts from the UML profile is shown in Fig. 2.

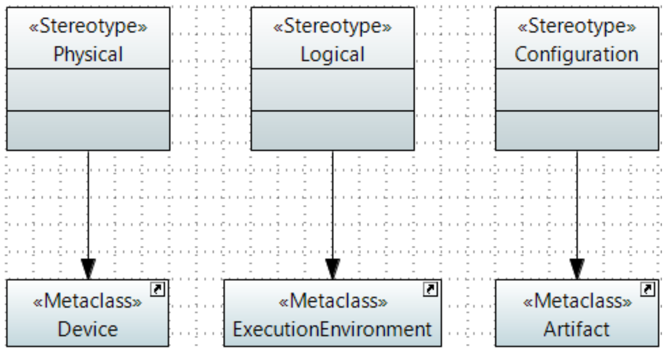
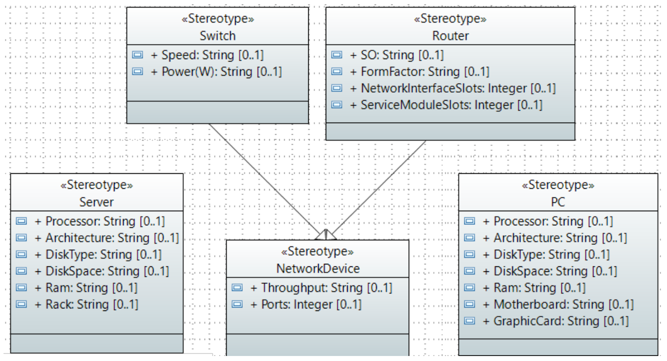
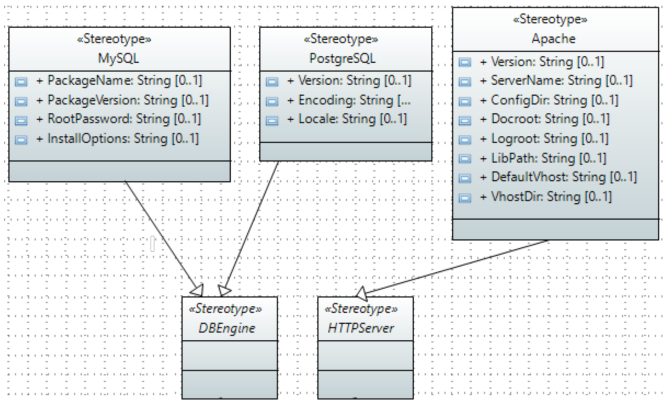


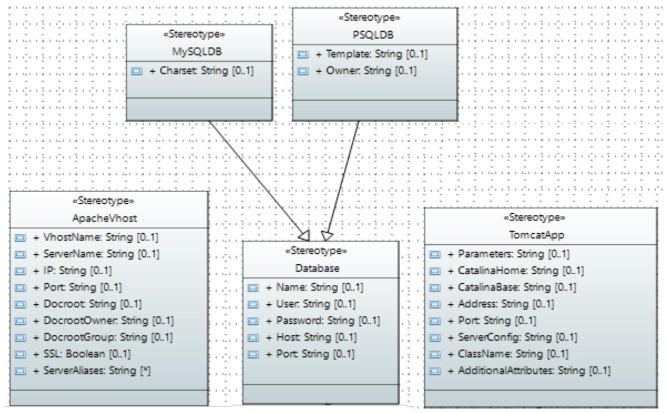
Figure 2: Physical, Logical and Configuration stereotypes


«Stereotype»
Switch
+ Speed: String [0..1]
+ Power(W): String [0..1]
«Stereotype»
Router
+ SO: String [0..1]
+ FormFactor: String [0..1]
+ NetworkInterfaceSlots: Integer [0..1]
+ ServiceModuleSlots: Integer [0..1]
«Stereotype»
Server
+ Processor: String [0..1]
+ Architecture: String [0..1]
+ DiskType: String [0..1]
+ DiskSpace: String [0..1]
+ Ram: String [0..1]
+ Rack: String [0..1]
«Stereotype»
NetworkDevice
+ Throughput: String [0..1]
+ Ports: Integer [0..1]
«Stereotype»
PC
+ Processor: String [0..1]
+ Architecture: String [0..1]
+ DiskType: String [0..1]
+ DiskSpace: String [0..1]
+ Ram: String [0..1]
+ Motherboard: String [0..1]
+ GraphicCard: String [0..1]

(a) Physical nodes (extends `Physical` from Fig. 2)



(b) Logical nodes (extends `Logical` from Fig. 2)


«Stereotype»
MySQLDB
+ Charset: String [0..1]
«Stereotype»
PSQLDB
+ Template: String [0..1]
+ Owner: String [0..1]
«Stereotype»
ApacheVhost
+ VhostName: String [0..1]
+ ServerName: String [0..1]
+ IP: String [0..1]
+ Port: String [0..1]
+ Docroot: String [0..1]
+ DocrootOwner: String [0..1]
+ DocrootGroup: String [0..1]
+ SSL: Boolean [0..1]
+ ServerAliases: String [*]
«Stereotype»
Database
+ Name: String [0..1]
+ User: String [0..1]
+ Password: String [0..1]
+ Host: String [0..1]
+ Port: String [0..1]
«Stereotype»
TomcatApp
+ Parameters: String [0..1]
+ CatalinaHome: String [0..1]
+ CatalinaBase: String [0..1]
+ Address: String [0..1]
+ Port: String [0..1]
+ ServerConfig: String [0..1]
+ ClassName: String [0..1]
+ AdditionalAttributes: String [0..1]

(c) Specific configurations (extends `Configuration` from Fig. 2)

Figure 3: Stereotype definitions (excerpt)

Then for each defined element within the three main concepts of `Physical`, `Logical` and `Configuration` we included them into the metamodel as specialization of the parent one. Due to space reasons, in Fig. 3 we only include a couple of examples of each kind.

We implemented this extension as an Eclipse plug-in, based on Papyrus[4], an Eclipse-based UML modeling tool which already has extension capabilities.

[4]Papyrus Modelling Environment: https://www.eclipse.org/papyrus/

B. Generation of configuration scripts

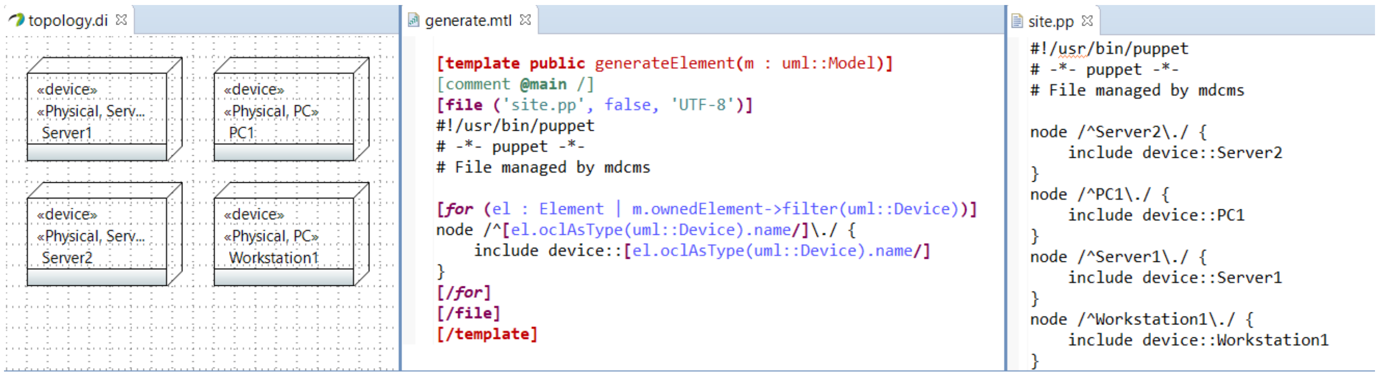
We use an existing CMT as a target of the configuration scripts generated, instead of using yet another new language, since generating for a specific tool will allow us to validate the scripts. To select the CMT we review several existing ones, from which we took an in-depth look at Puppet, Chef, and Ansible, since they stood up from the rest, selecting Puppet since we already worked with it. The solution we present in this subsection satisfies requirements RQ3, RQ5, and RQ6.

1) Mappings for the generation: We defined mappings between the network and configuration profile we have defined and the configuration scripts to be generated. For each element in the specific domain, we generate a file, and these files are organized in a directory hierarchy which is consistent with the target CMT, so it can be used to apply the configuration. The mappings are the following ones:

- two main folders are defined at root level: `manifests` and `modules`, and within the last one, there are two folders, one that includes the information of the physical and logical nodes called `device`, and another that corresponds to the configuration of these nodes, called `configurations`. These directories have their own `manifests` directory (this is exemplified in Fig. 7).
- a `site.pp` file (within the `manifests` folder, is generated with a list of the physical nodes to be configured (this is depicted in Fig. 4).
- for each *Physical node* a file is generated with *includes* to the files generated for the logical nodes belonging to it, under the `device` directory. Also, a directory is created for each `Physical node` which will contain the files for the corresponding `Logical nodes`.
- for each *Logical node* a file is generated with basic values regarding its installation, and *includes* to the configuration files generated for it, under the directory created for the `Physical node`. A directory is also created for each `Logical node` in the `configurations` directory, which will contain its configuration scripts.
- for each configuration, a file is generated and located in the directory of the `Logical node`.
- for any element for which does not correspond to create a configuration script, a text file with its information is created and located in the directory of the corresponding `Physical node` in a `Information` directory.

2) M2T transformation: The **mdcms2puppet** transformation takes as input a model defined with the UML profile and generates the configuration scripts in the Puppet format file, which will be imported in Puppet to configure the network. The transformation was implemented using Acceleo and integrated with the modeling tool. Complete information can be found at [9].

The output of the transformation is organized in the directory hierarchy we described, generating also a Puppet file that includes the modules to be installed, and the file `site.pp` (manifests/site.pp) which includes information about nodes in the topology, to be used by Puppet to declare nodes



```
[template public generateElement(m : uml::Model)]
[comment @main /]
[file ('site.pp', false, 'UTF-8')]
#!/usr/bin/puppet
# -*- puppet -*-
# File managed by mdcms

[for (el : Element | m.ownedElement->filter(uml::Device))]
node /^[el.oclAsType(uml::Device).name/]\./ {
    include device::[el.oclAsType(uml::Device).name/]
}
[/for]
[/file]
[/template]
```

```
#!/usr/bin/puppet
# -*- puppet -*-
# File managed by mdcms

node /^Server2\./ {
    include device::Server2
}
node /^PC1\./ {
    include device::PC1
}
node /^Server1\./ {
    include device::Server1
}
node /^Workstation1\./ {
    include device::Workstation1
}
```

Figure 4: Example of a source model, the M2T transformation and the file generated

and their configuration. This file is the starting point of the configuration, information regarding each node, either physical or logical, will be generated under **modules/device/manifests**, and configuration elements will be generated under **modules/configurations/manifests**.

Basic knowledge of Puppet is still needed to be able to make use of the generated code, is that it is necessary to run it in each one of the components that need to be configured. However, since a model contains all the information of the infrastructure, it could be possible to include the execution of the scripts as part of the generation process (or later on).

C. *Final Remarks*

The language covers most types of regular hardware and software components within a network for which a configuration is needed. There are many possible improvements, as discussed in Section VI. Also, the profile design could be improved to be more strict enforcing correct by construction modeling with strong typing of properties.

Unlike Puppet, our approach provides a generic approach based on topology models and a corresponding graphical notation, both aspects provide some benefits. First, the language is independent of any specific CMT, which seems to be more interesting for a heterogeneous IoT environment. In fact, Puppet is used just as a proof of concepts. Second, a graphic notation favors understanding when it comes to aspects of a topology, which Puppet does not represent. Nevertheless, the visualization we provide is only a default concrete syntax provided for deployment diagrams. Future work could be to explore other advanced and scalable alternatives.

V. Case Study

The case study validates the approach for basic system administration: unification of configuration and network elements, topology visualization and automatic configuration. However, as will be discussed in Section VI the whole approach can be extended in a more general context of IoT.

For space reasons we present an excerpt of the case study. The whole case study (available at [9]) comprises two routers connected to each other. On the one hand there is a subnet corresponding to the PCs and, on the other, there are the servers. Within the PC's subnet, depicted in Fig. 5, there are two switches (`SwitchWin` and `SwitchUnix`) that separate the Windows-based computers from Unix-based computers (and a printer `Printer`). Notice how the different devices are represented with their corresponding stereotypes. In particular, `PCUnix2` is a PC (a physical device) with two logical components installed within: `Ubuntu`, a logical operative system, and `mydb2`, a logical PostgreSQL db engine. There is also a configuration `PCUnix2DB` which is a `«PSQLDB»` configuration applied to the logical node `mydb2`.

The visualization of the network is partitioned: its topology together with an identification of the hardware, software and configurations within are depicted using deployment diagrams, as well as specific properties of their components are visualized in a properties view, as depicted in Fig. 6.

The generation process takes the network configuration model and generates every configuration file in a concrete directory structure (as depicted in Fig. 7) required by Puppet. A `Puppetfile` is generated containing the modules that need to be installed for use in Puppet, and also a `site.pp` file with the information of the existing nodes in the topology. If the configuration of the network is defined correctly, the generated files will be executable without modifications.

Within the `modules/device` directory there is a configuration file for each configurable device component. In the example, we have a directory for the `PCUnix2` physical device, and a Puppet file `PCUnix2.pp` with its configuration defined in Fig. 6a. The generated file is depicted in Fig. 8a. Within this directory, there is a Puppet file for each logical node, e.g., `mydb2.pp` depicted in Fig. 8b which was generated from the configuration in Fig. 6b.

Within the `modules/configurations` directory there is another directory for each component class, e.g., `db` for databases and the corresponding files generated from the configuration artifacts, e.g., `PCUnix2DB` configuration for the `mydb2` database. The corresponding Puppet file for this configuration is depicted in Fig. 8c. Notice that within the configuration file for the `mydb2` database (depicted in Fig. 8b) there is an import of the `PCUnix2DB` configuration.

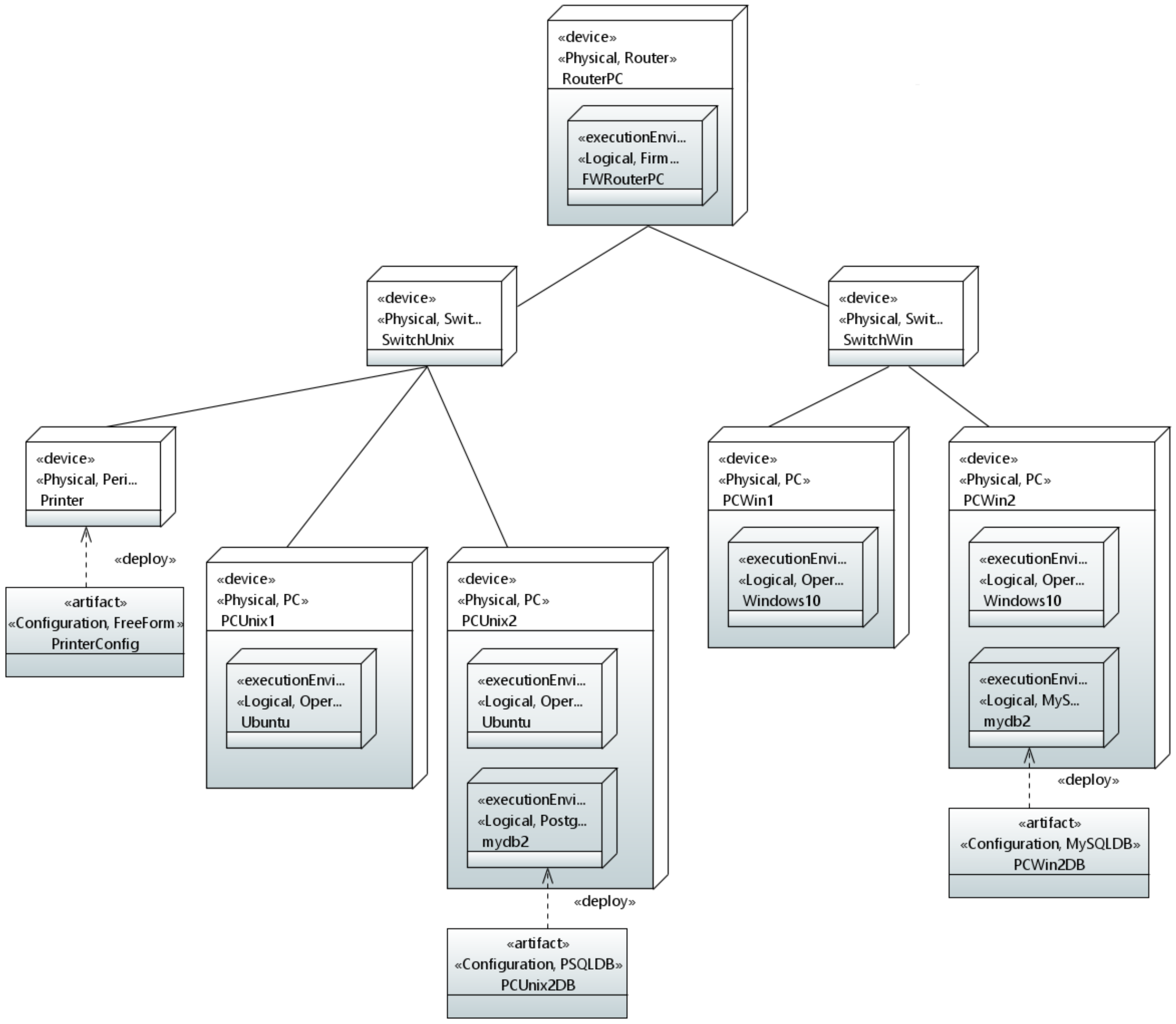


Figure 5: Case Study: network configuration

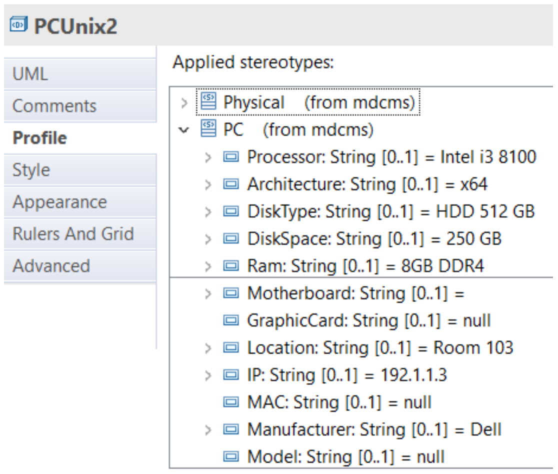
VI. Towards IoT Configuration Management

IoT poses a more general context than what we have presented, with a wide range of different devices and interconnection requirements. Our proposal can be considered the first step towards this context, at least for some IoT scenarios. As reviewed in [6], there are specific challenges for MDE, and from a CM perspective, we can discuss the following aspects.

1) Configuration and Control: Devices are usually provided with generic configurations which require to be further refined with specific settings. It is also required to remotely configure the devices for recovering and maintenance issues. As claimed in Section IV-C, it is useful to have a more abstract language, such as the one proposed, which links devices together with configurations, that can also be easily applied to many of them. The aid of a CMT is also beneficial, especially when it is possible to automatically generate scripts.

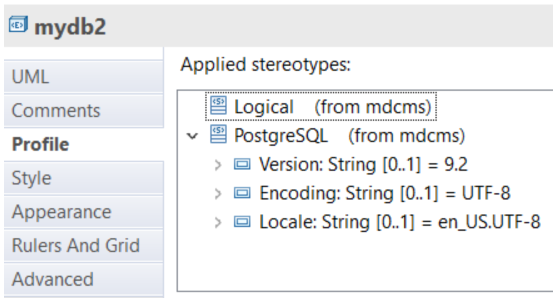
Heterogeneity is also an issue. Up to now, some technical aspects are fixed in the metamodel, e.g., concrete types of databases, so the language itself needs to change for adding new technologies. To improve language evolution, it could be interesting to essay a multi-level approach, such as in [16], providing means for metamodel extensibility.

We also claimed in Section III (RQ2) that we need to provide a common way of modeling both hardware and software aspects to improve communication and interoperability. We need to analyze how to relate our work with the Modeling and Analysis of Real-time and Embedded systems (MARTE) [17] proposal since it provides a more specific view on the specification, design, and verification/validation aspects of this kind of systems. Also, the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), a domain-specific language for defining portable deployment and automated management of services on a wide variety of infrastructure platforms, was analyzed for the automatic deployment of IoT environments [18]. Like ours, it also provides topology models and a corresponding graphical notation.



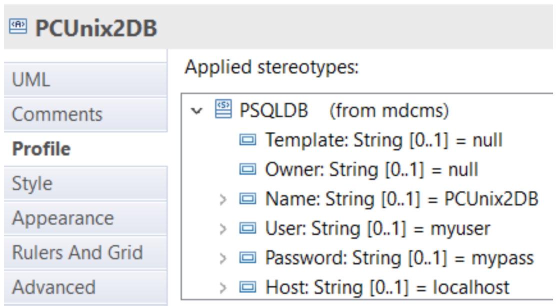
```
Applied stereotypes:
> Physical   (from mdcms)
v PC   (from mdcms)
    > Processor: String [0..1] = Intel i3 8100
    > Architecture: String [0..1] = x64
    > DiskType: String [0..1] = HDD 512 GB
    > DiskSpace: String [0..1] = 250 GB
    > Ram: String [0..1] = 8GB DDR4
    > Motherboard: String [0..1] =
      GraphicCard: String [0..1] = null
    > Location: String [0..1] = Room 103
    > IP: String [0..1] = 192.1.1.3
      MAC: String [0..1] = null
    > Manufacturer: String [0..1] = Dell
      Model: String [0..1] = null
```

(a) `PCUnix2` properties



```
Applied stereotypes:
  Logical   (from mdcms)
v PostgreSQL   (from mdcms)
    > Version: String [0..1] = 9.2
    > Encoding: String [0..1] = UTF-8
    > Locale: String [0..1] = en_US.UTF-8
```

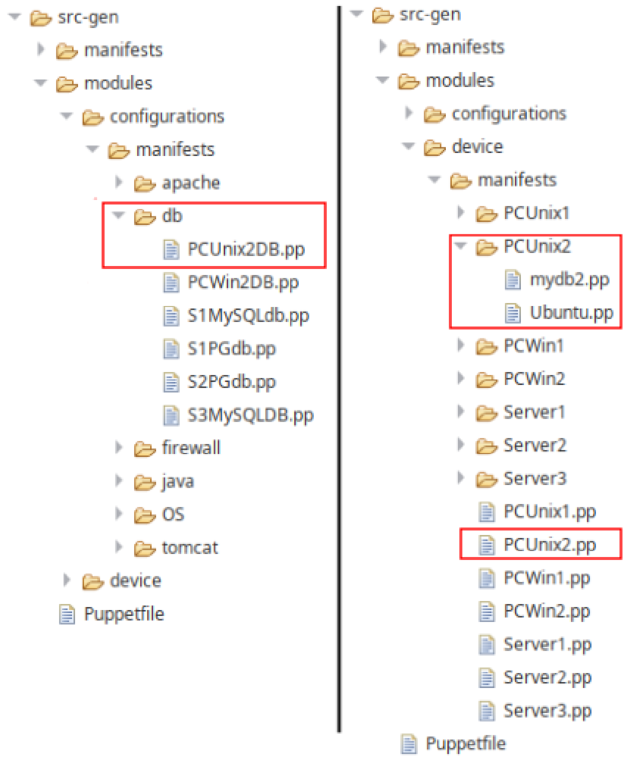
(b) `mydb2` properties



```
Applied stereotypes:
v PSQLDB   (from mdcms)
      Template: String [0..1] = null
      Owner: String [0..1] = null
    > Name: String [0..1] = PCUnix2DB
    > User: String [0..1] = myuser
    > Password: String [0..1] = mypass
    > Host: String [0..1] = localhost
```

(c) `PCUnix2DB` properties

Figure 6: Case Study: properties configuration



```
src-gen
  manifests
  modules
    configurations
      manifests
        apache
        db
          PCUnix2DB.pp
          PCWin2DB.pp
          S1MySQLdb.pp
          S1PGdb.pp
          S2PGdb.pp
          S3MySQLDB.pp
        firewall
        java
        OS
        tomcat
    device
  Puppetfile
```

```
src-gen
  manifests
  modules
    configurations
    device
      manifests
        PCUnix1
        PCUnix2
          mydb2.pp
          Ubuntu.pp
        PCWin1
        PCWin2
        Server1
        Server2
        Server3
        PCUnix1.pp
        PCUnix2.pp
        PCWin1.pp
        PCWin2.pp
        Server1.pp
        Server2.pp
        Server3.pp
  Puppetfile
```

Figure 7: Case Study: configuration files organization

2) Discovery and Monitoring: We already claimed, in Section III, that desirable requirement are the discovery of a network configuration (RQ7) and the monitoring of such configuration to automatically detect changes (RQ8). It could be useful to dynamically discover newly added devices and service processes. As presented in Section II, tools such as SolarWinds help in this direction. In previous work [19] we have dealt with the automated generation of Quality of Service (QoS) configurations for software services specified in the SoaML UML profile [20] which we can also integrate for a complete view of IoT services monitoring.

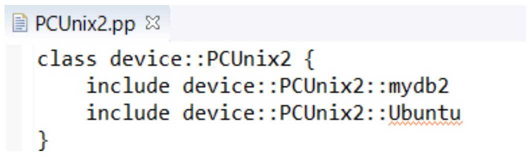
Concerning visualization, our language can be improved to support a more succinct concrete syntax in the way of existing network visualization tools. As discussed in Section IV-C we used the basic concrete syntax provided by deployment diagrams, but other notations could be considered to balance understanding and scalability.

3) Maintenance and Adaptation: Device heterogeneity poses a challenge for maintenance since not every device can be configured using the same CMT, e.g.: resource-constrained devices cannot execute a CMT client. As claimed before, it is convenient to generate different target languages from a more abstract model, even with uncertainty aspects (e.g., partial configuration). Cloud services and middleware typically play an important role. Several proposals can be integrated into this context, e.g., WSO2 Application Server Puppet Module [5].

Adaptation requirements are related to the models@runtime approach [21] which extends the applicability of models produced in MDE approaches to the runtime environment. In particular, sometimes IoT devices are required to adapt their behavior at runtime with little or no human intervention. Further work is needed in this sense.

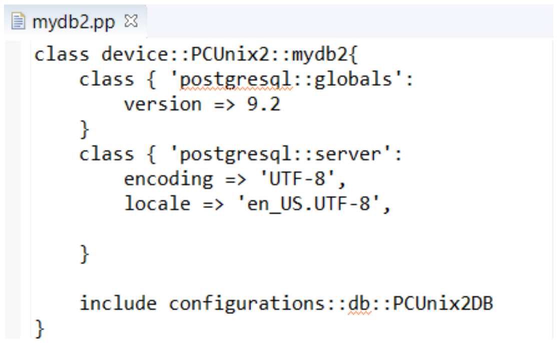
4) Security: As a final issue concerning configuration management, the administrator may meet problems of access control trying to reach a good balance between smooth execution and the proper protection for security and privacy. In this sense, it could be interesting to consider other models, such as SecureUML [22] which provides a modeling language for the development of secure, distributed systems based on Role-Based Access Control (RBAC) with additional support for specifying authorization constraints.

[5]WSO2 App Server Puppet Module: https://github.com/wso2/puppet-iot


PCUnix2.pp

```
class device::PCUnix2 {
    include device::PCUnix2::mydb2
    include device::PCUnix2::Ubuntu
}
```

(a) PCUnix2 generated script

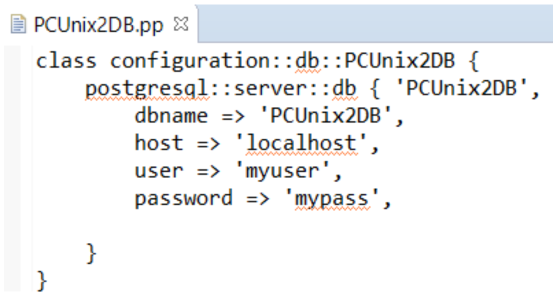

mydb2.pp

```
class device::PCUnix2::mydb2{
   class { 'postgresql::globals':
      version => 9.2
   }
   class { 'postgresql::server':
      encoding => 'UTF-8',
      locale => 'en_US.UTF-8',

   }

   include configurations::db::PCUnix2DB
}
```

(b) mydb2 generated script


PCUnix2DB.pp

```
class configuration::db::PCUnix2DB {
    postgresql::server::db { 'PCUnix2DB',
        dbname => 'PCUnix2DB',
        host => 'localhost',
        user => 'myuser',
        password => 'mypass',

    }
}
```

(c) PCUnix2DB generated script

Figure 8: Case Study: generated configuration scripts

VII. Conclusions & Future Work

We experimented with a model-driven approach for system administration linking visualization and automation needs. A domain-specific language (a UML profile of deployment diagrams) links hardware, software and configuration aspects of network infrastructure, and a model transformation generates Puppet configuration scripts based on these models.

A common language for the whole network configuration simplifies the communication between members of a team, which reduces the problems associated with it. Moreover, the use of a visual editor and a language with known semantics (UML deployment diagrams) considerably reduces the access barrier to the use of configuration management systems. The proposal is currently being evaluated in a real context associated with the management of the basic infrastructure configuration for electronic government.

Although it is as exploratory work, it lays the foundations for its application in a more general context of IoT. In this context, there are many open issues for its improvement, as presented in Sections III and VI, ranging from the extension of the language with the support of new components in an IoT environment and the definition of transformations to other configuration management tools and platforms, to the improvement of the tool with respect to its visualization features and the automatic discovery of the network configuration.

References

[1] T. Limoncelli, C. Hogan, and S. Chalup, *The Practice of System and Network Administration*. Addison-Wesley Professional, 2007.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[4] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[5] J. Quigley and K. Robertson, *Configuration Management: Theory, Practice, and Application*. Addison-Wesley Professional, 2003.

[6] S. Wolny, A. Mazak, and B. Wally, "An initial mapping study on mde4iot," in *Proc. of MODELS 2018 Workshop: MDE4IoT*, ser. CEUR Workshop Proceedings, vol. 2245. CEUR-WS.org, 2018, pp. 524–529.

[7] S. Kent, "Model driven engineering," in *International Conference on Integrated Formal Methods*. Springer, 2002.

[8] F. Research, "The forrester wave: Configuration management software for infrastructure automation, q4 2018," 2018.

[9] M. Centurion, M. Kotvinsky, D. Calegari, and A. Delgado, "MdNetConf: Model-driven Network Configuration software," 2019. [Online]. Available: https://gitlab.fing.edu.uy/open-coal/mdnetconf

[10] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs. structural," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 147–159.

[11] F. Essaadi, Y. B. Maissa, and M. Dahchour, "Mde-based languages for wireless sensor networks modeling: A systematic mapping study," in *Advances in Ubiquitous Networking 2*. Springer, 2017, pp. 331–346.

[12] M. Bjorklund, "Yang-a data modeling language for the network configuration protocol," Internet Engineering Task Force, Standard, 2010.

[13] G. A. H. Castro, "Metamodelo para configuraciones en dispositivos de redes como estándar soportado en la ingeniería dirigida por modelos," Master's thesis, Universidad "Francisco José de Caldas", 2016.

[14] T. Buchmann, A. Dotor, and B. Westfechtel, "Model-driven development of software configuration management systems - A case study in model-driven engineering," in *Proc. of the 4th Intl. Conf. on Software and Data Technologies (ICSOFT), Volume 1*. INSTICC Press, 2009, pp. 309–316.

[15] H. Giese, A. Seibel, and T. Vogel, "A model-driven configuration management system for advanced it service management," in *Proc. of 4th Intl. Workshop on Models@run.time*, vol. 509, 10 2009, pp. 61–70.

[16] S. Jácome and J. de Lara, "Controlling meta-model extensibility in model-driven engineering," *IEEE Access*, vol. 6, pp. 19 923–19 939, 2018.

[17] Object Management Group, "A uml profile for marte: Modeling and analysis of real-time embedded systems," 2015. [Online]. Available: https://www.omg.org/omgmarte/

[18] A. C. F. da Silva, U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, B. Mitschang, and R. Steinke, "Internet of things out of the box: Using TOSCA for automating the deployment of iot environments," in *Proc. of 7th Intl. Conf. CLOSER*. SciTePress, 2017, pp. 330–339.

[19] A. Delgado, "Qos modeling and automatic generation from soaml service models for business process execution," in *2015 IEEE International Conference on Services Computing*, 2015, pp. 522–529.

[20] A. Delgado and L. González, "Eclipse SoaML: A tool for engineering service oriented applications," in *Joint Proc. of the CAiSE 2014 Forum*, ser. CEUR Workshop Proceedings, vol. 1164. CEUR-WS.org, 2014, pp. 201–208.

[21] G. S. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[22] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A uml-based modeling language for model-driven security," in *Proc. of Intl. Conf. on the Unified Modeling Language*. Springer, 2002, pp. 426–441.